DrPIM: An Adaptive and Less-blocking Data Replication Framework for Processing-in-Memory Architecture

Sheng Xu
School of Computer and Information,
Anhui Normal University,
Wuhu, China
Institute of Artificial Intelligence,
Hefei Comprehensive National Science Center,
Hefei, China
xusheng2019@ahnu.edu.cn

Hongyu Xue
School of Computer and Information,
Anhui Normal University,
Wuhu, China
adiwenxue@ahnu.edu.cn

Le Luo
School of Computer and Information,
Anhui Normal University,
Wuhu, China
2020020@ahnu.edu.cn

Liang Yan
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
yanliang19b@ict.ac.cn

Xingqi Zou
Institute of Computing Technology,
Chinese Academy of Sciences
Beijing, China
zouxingqi@ict.ac.cn

ABSTRACT

Processing-in-Memory (PIM) architecture suffers from frequent data synchronization, which raises inevitable coherent stalls and consumes more energy. This paper presents *DrPIM*, an adaptive and less-blocking data replication framework for Processing-in-Memory architecture to address such issues. The key insights are 1) finer-grained data management for the conflict data region and 2) the automatic generation and invalidations of data replications. The above two schemes significantly decrease the critical path of data synchronization and the access latency for conflict data, thus providing a less-blocking execution. Evaluations show that DrPIM achieves a speedup of 1.5x over the state-of-the-art and reduces the data-moving energy by 49%.

CCS CONCEPTS

• **Computer systems organization → Architectures.**

KEYWORDS

Processing-in-Memory, data replication, coherence

ACM Reference format:

Sheng Xu, Hongyu Xue, Le Luo, Liang Yan, and Xingqi Zou. 2023. DrPIM: An Adaptive and Less-blocking Data Replication Framework for Processing-in-Memory Architecture. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI'23). June 5-7, Knoxville, TN, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/ 3583781.3590294

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA.

ACM 979-8-4007-0125-2/23/06
DOI: https://doi.org/10.1145/3583781.3590294

1 INTRODUCTION

PIM is considered a promising technique to hit the memory wall. Despite the significant benefits of PIM, data coherence and synchronization are still the main challenges in PIM systems [1]. Experiments showed that an ideal PIM system, i.e., a PIM system without data coherence penalty, can double the performance of the state-of-the-art PIM designs [2].

Prior works [2, 3, 4] have proposed several designs to alleviate the overhead of data coherence and synchronization in PIM. Some works migrate a conventional shared memory model, allowing a global directory-based lock to provide exclusive access to the same data region. Traditional coherence mechanisms, e.g., MESI and MOESI, are extended and applied. However, such schemes are impractical as massive broadcast traffic travels through a narrow off-chip link, which brings potential damage to the system. To minimize the coherent traffic caused by cache-line updates, many others provide dedicated message-passing schemes by designing specified programming models, data placement, and structures. Although well-designed, these approaches leave efforts to the programmers, resulting in high programming complexity. For instance, *SynCron* [3] adds low-cost hardware to support efficient PIM synchronization but requires manual shared data lock management like *lock_acquire()* and *lock_release()*. To further reduce the off-chip synchronization and get compatible with the existing architecture, others [2] perform a coarse-grained coherence which further enlarges the coherent granularity to a page or a bank [4]. As a result, even though a small portion of the data is accessed, the whole data region is flush to the memory, adding more latency and energy consumption in some cases [5].

To overcome the challenge of PIM coherence without extra programming complexity, we propose *DrPIM*, an adaptive and less-blocking data replication framework for PIM architecture. DrPIM provides finer-grained data management for the conflict data region and automatically generates data replications to decrease the critical path of data synchronization. DrPIM is transparent and works well with the existing programming model and requires no modifications to the source code.

2 DATA REPLICATION IN PIM

To understand why PIM can benefit from data replication schemes, we need first to quantify the impact of replica placement on the system. Our target system is a PIM system with multiple HMCs organized in a mesh topology like Fig. 1. It performs a lock-based coarse-grained coherence under the granularity of a page. In PIM systems with HMCs, data synchronization latency and energy are demonstrated by 1) the distance between data residence and execution location, and 2) coherent stalls caused by the spin-lock feature. If data replication is applied in the PIM system, shared data could be distributed in multiple cubes or vaults. In-memory logic could access the data from the nearest replicas for a shorter distance to reduce access latency and data-moving energy. Besides, data replication may provide lock-free data access to alleviate the coherent stalls. However, only *read* can benefit from data replications, as *write* will destroy any existing replicas of the same data region in the memory. Therefore, the benefit of data replication is closely related to the remote read pattern of the applications. To make an observation of the read pattern in PIM, use the following terminology:

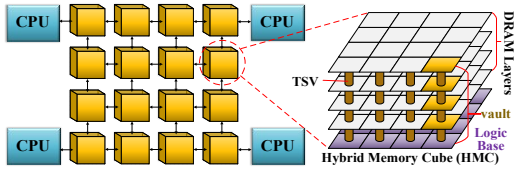

Figure 1: An illustration of the HMC-based PIM system. HMCs are organized in a mesh topology and interconnected through off-chip links.

Center of Read Access (C_{AR}): a C_{AR} of a page refers to the equivalent position to store the page where all reads to the page have the shortest accessing distance. In a mesh HMC topology, C_{AR} can be defined as a coordinate:

$$C_{AR} = \left(\frac{\sum_{i \in HMCs} X_i N_i}{\sum_{i \in HMCs} N_i}, \frac{\sum_{i \in HMCs} Y_i N_i}{\sum_{i \in HMCs} N_i} \right) \quad (1)$$

in which $HMCs$ is the set of all cubes, X_i and Y_i are the x and y coordinate of the cube i. N_i is the read amount of read access to the page of the cube i.

The Average Read Manhattan Distance (D_{AR}): a D_{AR} of a page refers to the average read distance to access the page, which is expressed in the inter-cube hops. It is obvious that D_{AR} is the Manhattan distance to the C_{AR} of the page. D_{AR} can be defined as:

$$D_{AR} = \frac{\sum_{i \in HMCs} (|X_i - C_{ARx}| + |Y_i - C_{ARy}|) N_i}{\sum_{i \in HMCs} N_i} \quad (2)$$

in which C_{ARx} and C_{ARy} are the x and y coordinate of C_{AR}.

Fig. 2a shows the distribution of remote read requests to the **locked** pages from the in-memory logic derived from *PageRank* using the *soc-LiveJournal1* dataset in an 8x8 mesh PIM system. The overall C_{AR} and D_{AR} can be computed using equations (1) and (2). As depicted in Fig. 2b, most of the C_{AR} are located near the cube with the coordinates (2, 3) (denoted as •) and the average D_{AR} is about 4.14, which indicates that each remote read to the original page (*data owner*, denoted as blue •) has to wait for page unlocking and then travels through more than four HMCs on average. However, suppose we create two replicas of the page (denoted as red •) and perform sub-regional coherence independently. In that case, nearby in-memory logic can benefit from a shorter distance and less coherent stalls.

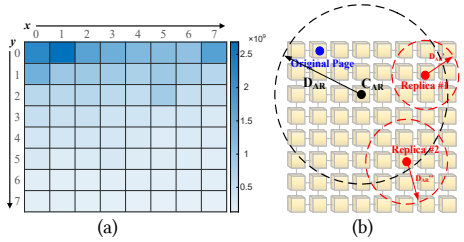

Figure 2: a) distribution of remote read requests to the locked pages from all in-memory logic for *PageRank* using the *soc-LiveJournal1* dataset. b) An example of replica generation.

3 DRPIM FRAMEWORK

DrPIM includes three steps: 1) a replica generation scheme that finds the optimal pages, location, and the number of replicas to minimize the critical path of data synchronization, 2) a replica invalidation approach to minimize the overhead caused by inefficient replicas placement, and 3) a synchronization mechanism to prevent replicas from being inconsistent with others. The above three steps must be executed periodically at run-time to invalidate any out-of-date replicas and reproduce new ones. In the following of this paper, we use the term ***epoch*** to denote the time interval to re-think the effectiveness of existing replicas in the memory.

3.1 Replica Generation

To perform data replication and gain enough benefits, DrPIM performs three phases: First, DrPIM selects the candidate pages that may benefit from data replication. Second, DrPIM establishes a benefits evaluation to determine if a candidate page should be dispatched and is ready for data replication. Third, DrPIM calculates the optimal location and the number of replicas for each dispatched page and then generates the replicas for them.

Candidate Page Identification Phase. To identify the optimal pages that may benefit from replication, the most accurate way is to track every remote access (both *reads* and *writes*) and maintain an access list. However, such implementation introduces non-trivial overheads since it adds unneglected bits per page. To keep the overhead low, DrPIM only records the pages within the most significantly accessed bank (denoted as $Bank_i$) in the last epoch and uses that to identify candidate pages. $Bank_i$ can be easily captured by snooping on the broadcast traffic in the vault logic.

To pick up candidate pages inside $Bank_i$, more definitions are needed. We use the term *Read Ratio*, denoted as r_p, to represent the percentage of remote reads in all remote accesses to a page. *Read Ratio* can be defined as:

$$r_p = \frac{N_p}{\sum_{j \in HMCs} N_j} \quad (3)$$

Since only remote reads could benefit from data replication, *Read Ratio* can partly represent if a shared page was frequently accessed during its locked-in time. Therefore, DrPIM only picks up candidate pages from $Bank_i$ that 1) have the top N read ratio and 2) exceed a pre-defined threshold T_{rr}. The first condition ensures that DrPIM only selects frequently read pages, and the second prevents DrPIM from selecting pages that are frequently updated, which may affect the efficacy of DrPIM. The determination of the parameter N is dataset-related. In our experiments, N can be very small in most cases.

Page Dispatching Phase. In this phase, DrPIM establishes a profit function to estimate the benefits of replicating a candidate page, which is defined as:

$$Profit_p = (\alpha D_{ARp} + \beta r_p) \times N_p \geq T_{cost} \quad (4)$$

where D_{ARp}, r_p, and N_p are the average read Manhattan distance, *Read Ratio*, and the remote read amount of candidate page p, α and β are relative weights. Equation (4) reflects the benefits of data replication come from two aspects: First, the more access distance is (that is, $D_{ARp} \times N_p$), the more benefits from replication, since data replication can reduce D_{AR} for the page, which reduces the data synchronization energy. Second, the larger the *Read Ratio* is (that is, $r_p \times N_p$), the fewer stalls caused by coherence. Besides, DrPIM only dispatches the candidate page when the profit function exceeds a fixed threshold, T_{cost}. When candidate pages are satisfied with Equation (4), DrPIM dispatches the page for the next phase.

Replica Generation Phase. Upon a page having been dispatched for replication, DrPIM determines the optimal location and the number of copies of the page. We propose a regional-based algorithm to make such a decision. The main idea of this algorithm is to separate all HMCs into four sub-regions recursively and evaluate the benefit of generating replicas in the sub-region, which is shown in Algorithm 1. It has the following three major steps. First, as for an HMC interconnection topology graph G, we generate C_{AR} and D_{AR} for a dispatched page p, using the remote read matrix M monitored by the memory controller of each vault in the last epoch (lines 1-2). If the region has no more than two HMC cubes the algorithm is completed (lines 3-5), as there is no need to generate replicas in other vaults of the same cube. If not, we partition the region into four sub-regions (line 6). For instance, for an 8x8 mesh HMCs system with $C_{AR} = (2,3)$ and $D_{AR} \approx 4.14$, the algorithm partitions the region into four groups: {(0~2, 0~3), (0~2, 4~7), (3~7, 0~3), (3~7, 4~7)}. Then a loop is executed until all the sub-regions are traversed (line 7). Next, we compute the C_{AR} and D_{AR} of the sub-region for p (lines 8-10), and use the following benefit equation to evaluate the profit to create a replica in the sub-region (line 11):

$$Replica_p = N_p r_p (D_{AR} - D_{ARsub}) - N_p (1 - r_p)(|x - x_{sub}| + |y - y_{sub}|) > T_{rep} \quad (5)$$

where N_p is the remote read amount of the sub-region G_{sub}. $N_p r_p (D_{AR} - D_{ARsub})$ refers to the distance save of all remote read in G_{sub} if the replica is generated; $(|x - x_{sub}| + |y - y_{sub}|)$ stands for the average Manhattan distance from global C_{AR} to regional C_{ARsub}. $N_p (1 - r_p)(|x - x_{sub}| + |y - y_{sub}|)$ represents the synchronization overhead to establish the replica. We also add a threshold, T_{rep}, to make the evaluation sense (line 11). If T_{rep} is reached, DrPIM adds C_{ARsub} to the replica generation lists and performs this algorithm for the sub-region recursively to find if more replicas should be placed in the sub-region (lines 12-13). Once the position and the number of replicas are determined, DrPIM transfers the data to destiny instantly and broadcasts the replica generation messages as long as the data owner is still clean.

Algorithm 1 Replica Generation Algorithm

Input: HMC interconnection topology graph G; Remote read matrix M to a dispatched page p in the last epoch; Read ratio of the page r_p in the last epoch;

Output: The coordinate set of generated replicas $\{Rep\}$

$\{Rep\} \Leftarrow$ ***function*** $ReplicaGeneration(G, M, r_p)$

1: $(x, y) \leftarrow calculateC_{AR}(G, M)$
2: $D_{AR} \leftarrow calculateD_{AR}(G, M, x, y)$
3: if $numofCubes(G) < 2$ **then**
4: **return**
5: end if
6: $\{SubRegion\} = \{(0 \le x' \le \lfloor x \rfloor, 0 \le y' \le \lfloor y \rfloor), (0 \le x' \le \lfloor x \rfloor, \lfloor y \rfloor < y' < n), (\lfloor x \rfloor < x' < n, 0 \le y' \le \lfloor y \rfloor), (\lfloor x \rfloor < x' < n, \lfloor y \rfloor < y' < n)\}$
7: foreach $G_{sub} \in \{SubRegion\}$ **do**
8: $M_{sub} \leftarrow getSubRegionMatrix(M, G_{sub})$, $N_P = \sum M_{sub}$
9: $(x_{sub}, y_{sub}) \leftarrow calculateC_{AR}(G_{sub}, M_{sub})$
10: $D_{ARsub} \leftarrow calculateD_{AR}(G_{sub}, M_{sub}, x_{sub}, y_{sub})$
11: **if** $N_p r_p D_{AR} - N_p (1 - r_p)(|x - x_{sub}| + |y - y_{sub}|) > T_{rep}$ **do**
12: $\{Rep\} += (x_{sub}, y_{sub})$
13: $ReplicaGeneration(G_{sub}, M_{sub}, r_p)$
14: **end if**
15: end foreach

3.2 Replica Coherence and Synchronization

In DrPIM, we propose a regional finer-grained coherence scheme for data coherence and synchronization. For dispatched page and its replicas, DrPIM divides the shared page into K sub-regions and keeps the lock bit independently. The idea of adopting regional finer-grained coherence comes from [5], which suggests that only a few cache lines in each shared page are modified in PIM systems. Such implementation in DrPIM guarantees that conflict pages with replicas retain most of the fine-grained coherence's less-blocking benefits while the others can enjoy much less coherent traffic brought by coarse-grained coherence.

To perform regional finer-grained coherence, DrPIM maintains a directory-based table, denoted as *Replica Page Table* in the vault controller. Each entry of the *Replica Page Table* holds K valid bits

of each sub-region of the page, the original data's location, and the location of all replicas of the same page. Whenever a *write* request to the pages occurs, DrPIM first broadcasts a message to all other replicas of the same page and invalids the corresponding valid bit of the accessed sub-regions. Then DrPIM forwards the write request to the location where the original data resides (i.e., data owner in Fig. 2b) to apply data modifications. Only the data owner holds the dirty data of sub-regions. If a memory request accesses the invalidate sub-region of a replica (say, the sub-region has been modified), the request will also be forwarded to the data owner's location. This mechanism ensures that all modifications to pages with replicas can only be applied at the data owner while providing lock-free access for read requests to unmodified sub-regions.

3.3 Replica Invalidation

Since the memory access pattern changes throughout the application execution, the effectiveness of replicas should be re-evaluated periodically too. Removing inefficient replicas can free up memory space and decrease the synchronization overhead.

The replication invalidation in DrPIM is relatively simple: if a replica needs to be invalidated, it must meet at least one of the following conditions: 1) all sub-regions of the page are all invalid, 2) the replica is rarely read but written frequently, and 3) the replica page table is full meanwhile a newly generated replica evicts the entry.

Algorithm2 Replica Invalidation Algorithm

Input: Replica Page Table of data owner RPT_{owner} ;

```
1: foreach Entry_i ∈ RPT_owner do
2:     if {ValidBits} ∈ Entry_i is all invalidate do
3:         foreach Replica_j ∈ Entry_i do
4:             Invalidate(Replica_j)
5:         end foreach
6:         Remove(Entry_i)
7:     else
8:         foreach Replica_j ∈ Entry_i do
9:             RPT_j ← findRPTofReplica(Replica_j)
10:            N_valid, N_all ← getAccessCounter(RPT_j, Replica_j)
11:            if N_valid / N_all < T_v do
12:                Invalidate(Replica_j)
13:            end if
14:        end foreach
15:        if ReplicaCount(Entry_i) is 0 do
16:            Remove(Entry_i)
17:        end if
18:    end if
19: end foreach
```

As for Condition 1, DrPIM should invalidate the replicas of the page immediately as all data accesses to the page have to be forwarded to the data owner, which lessens the effectiveness of data replication. As for Condition 3, the replicas should also be invalidated immediately if the invalidation is caused by the *Replica Page Table*'s eviction since no inconsistency will occur. As for Condition 2, DrPIM uses the following algorithm to perform replication invalidation at the end of each epoch (see Algorithm 2). DrPIM first traverses through all entries of the data owner's *Replica Page Table* to find if any dispatched page has no unmodified sub-regions (lines 1-3). If so, it indicates that every access to the replicas of the page must be forwarded to the data owner (say, meet Condition 1). Therefore, DrPIM invalidates all replicas of the page and removes the corresponding entry in the *Replica Page Table* (lines 4-6). Else, DrPIM will iterate all the replicas of the page to check their effectiveness using the following equation (lines 8-10):

$$N_{valid} / N_{all} < T_v \tag{6}$$

where N_{valid} and N_{all} are the valid access amount and total access amount to the replica in the last epoch, thus N_{valid} / N_{all} indicates if the page can benefit from lock-free reads. The percentage should be lower than a threshold, T_v (lines 11-14). After performing invalidation, if there is no replica for a dispatched page, DrPIM will free up the corresponding entry.

4 EVALUATION

4.1 Experimental Setup

Simulation configurations: We use an in-house simulator that integrates ZSim [6] and HMCSim [7] to simulate the behavior of DrPIM. The detailed architecture parameters of DrPIM are listed in Table 1. We compare DrPIM with four schemes: 1) *CPU-Only*: a conventional architecture that employs HMCs as the main memory without any memory offloading. All the evaluated applications are executed with 32 threads. 2) A *Coarse-Grained* scheme enables the coarse-grained coherence of page granularity in PIM systems. 3) A *Message-Passing* scheme that performs like *TESSERACT* architecture. 4) An *Ideal* scheme without any synchronization latency and energy overhead.

Table 1: System Configuration

Component	Configuration
On-chip Processors	4 cores, OoO, 2GHz frequency L1 I/D-Caches: 32KB private, 4-way, 4-cycle latency, 64B line L2 Cache: 2MB shared, 8-way, 35-cycle latency, 64B line
Memory Model	64 HMCs organized in an 8×8 mesh (Fig. 3a) 8GB, 32 vaults per cube, 512 banks 32 single-issue in-order cores, 2GHz, each per cube L1 I/D-Caches: 32KB private, 4-way, 4-cycle latency, 64B line Bandwidth: TSV 10GB/s, 320GB per cube; 120GB/s per link; Energy: 48pJ/bit on link, 10.48pJ/bit intra-HMC [7]
DrPIM Parameters	$T_{rr}=0.7$, $\alpha=2$, $\beta=40$, $T_{cost}=1500$, $T_{rep}=300$ $T_v=0.6$, $N=16$, $K=8$ 30000 memory accesses each epoch

Workloads and datasets: We use some typical PIM applications, which are also used in [1-5] as listed in Table 2. All real-world datasets [8] used are statically partitioned across HMCs, where the vertex data is equally distributed across cubes.

Table 2: Workloads and Datasets

Workloads	Datasets
Average Teenage Follower (**ATF**), Breadth-First Search (**BFS**), PageRank (**PR**), Bellman-Ford Shortest Path (**SP**)	*com-youtube* (1.1M Vertices, 2.9M Edges), *wiki-talk* (2.3M Vertices, 5M Edges), *com-orkut* (3.1M Vertices, 117M Edges), *cit-Patents* (3.7M Vertices, 16.5M Edges), *com-lj* (4M Vertices, 34.7M Edges), *soc-LiveJournal1* (4.8M Vertices, 69M Edges)

4.2 Evaluation Results

Performance evaluation: Fig. 3 presents a performance comparison of the five configurations. All results are normalized to *CPU-Only*. DrPIM improves the average performance by 50% over

CPU-Only, and 49% over *Coarse-Grained*. To generate data replicas and perform multi-grained coherence for the dispatched pages, DrPIM retains most of *Ideal's* benefits, coming within on average 13.8% of the performance of *Ideal*.

Effectiveness of Replica Generation: We also evaluate the effectiveness of the entire replica generation framework. Fig. 4 shows the average number of candidate pages and the average number of replicas generated in the last epoch. DrPIM detects seven candidate pages per epoch on average, and 63.7% of them are dispatched for replica generation. In each epoch, DrPIM generates about 14 replicas to perform finer-grained coherence.

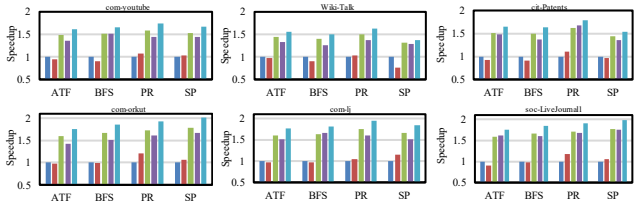

Figure 3: Normalized performance evaluation.

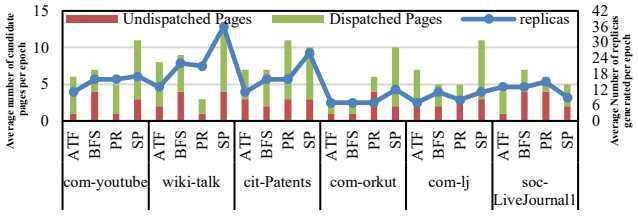

Figure 4: The number of generated replicas.

Data movement energy: Fig. 5 shows the normalized data movement energy evaluation for all schemes except *CPU-Only*. DrPIM reduces data-moving energy across all workloads by 1.16x and 1.94x over *Coarse-Grained* and *Message-Passing* schemes, respectively. During the execution, *Message-Passing* suffers from massive data synchronization, which causes significant data-moving energy. In contrast, DrPIM can benefit from multi-grained coherence, which reduces coherent conflicts while retaining the low coherent traffic.

4.3 Overhead

To identify candidate pages, DrPIM needs to track the banks with the most significant remote access. It can be implemented by simply maintaining an access counter for each bank. To pick up pages with the top N read ratio in the bank, DrPIM should hold a counter for each page in the bank. For an 8GB HMC, each bank has 16MB storage, that is 4096 pages per bank by default. To track the remote read amount of 4096 pages, it leads to an extra $\log_2(4096)\times4096=6$KB to each vault controller. Such overhead can be migrated by integrating the logic into the cache of the vault core, as the in-memory cache hit rate is relatively low.

5 CONCLUSIONS

In this paper, we propose DrPIM, an adaptive data replication framework. DrPIM automatically detects data regions that suffer from data synchronization. We also propose dedicated algorithms to create replicas at the optimal location and invalidate stale or defunct replicas. Regional finer-grained data management is applied in DrPIM to reduce the data access contention caused by the coarse-grained coherence. Experiment shows that DrPIM can achieve a speedup of 1.5x over the state-of-the-art and provide a less-blocking execution for PIM systems.

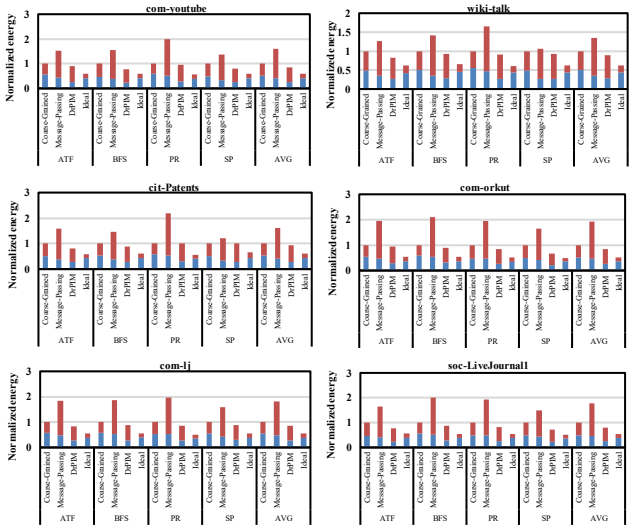

Figure 5: Normalized data movement energy evaluation.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (62102005, 62104230), Anhui Natural Science Foundation (2008085QF330, 2108085QF265), The University Synergy Innovation Program of Anhui Province (GXXT-2021-011), and the Research Program of Anhui Normal University (751968).

REFERENCES

[1] Li, Zerun, Xiaoming Chen, and Yinhe Han. "Optimal Data Allocation for Graph Processing in Processing-in-Memory Systems." *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022.

[2] Boroumand, Amirali, et al. "CoNDA: Efficient cache coherence support for near-data accelerators." *Proceedings of the 46th International Symposium on Computer Architecture*. 2019.

[3] Giannoula, Christina, et al. "Syncron: Efficient synchronization support for near-data-processing architectures." *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021.

[4] Xu, Sheng, et al. "CuckooPIM: an efficient and less-blocking coherence mechanism for processing-in-memory systems." *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 2019.

[5] Boroumand, Amirali, et al. "LazyPIM: Efficient support for cache coherence in processing-in-memory architectures." *arXiv preprint: 1706.03162* (2017).

[6] Sanchez, Daniel, and Christos Kozyrakis. "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems." *ACM SIGARCH Computer architecture news*. 41.3 (2013): 475-486.

[7] Leidel, John D., and Yong Chen. "Hmc-sim-2.0: A simulation platform for exploring custom memory cube operations." *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016.

[8] Leskovec, Jure, and Rok Sosič. "Snap: A general-purpose network analysis and graph-mining library." *ACM Transactions on Intelligent Systems and Technology (TIST)*. 8.1 (2016): 1-20.